Universiteit Leiden

# Master Computer Science

Analyzing and Simplifying Contrastive Reinforcement Learning

| | |
|---|---|
| Name: | Juri Morisse |
| Student ID: | s3737764 |
| Date: | 25/08/2024 |

Specialisation: Artificial Intelligence

| | |
|---|---|
| Supervisor: | Thomas Moerland |
| Supervisor: | Felix Kleuker |
| 2nd reader: | Álvaro Serra-Gómez |

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

# Abstract

Representation learning plays a crucial role in many machine learning tasks, including reinforcement learning (RL). Traditionally, representations are learned either end-to-end or in a decoupled fashion as part of a broader RL algorithm. This work builds on a novel approach introduced by Eysenbach et al. (2023) who propose to not learn representations as an ingredient for RL but, instead, learn representations that directly solve goal-conditioned RL tasks. Using contrastive representation learning on online collected trajectories, encoders are trained to produce representations that capture temporal correlation. By applying this approach to learn state-action and goal representations, representation similarities are trained to correspond to the likelihood of reaching the goal given the state-action pair. A critic defined as the inner product between state-action and goal representations, thus, learns to approximate a goal-conditioned Q-function and can be used to train a parameterized actor to solve goal-conditioned RL problems. This technique integrates RL and representation learning into a single objective and can learn in a self-supervised manner without the need of a reward function.

In our work, we perform an in-depth analysis of this approach which we call contrastive reinforcement learning (CRL). We visualize CRL's learned representation space and critic. Our visualizations support the claims made by CRL that representations capture the environment's actionable structure and can be used to approximate the likelihood of reaching future states. Further, we analyze the behavior of CRL's trained actor and find it not to align with a strategy of selecting actions greedily in regards to their critic value. Since our experiments also show that such a greedy strategy allows for equal performance, we propose to replace the original algorithm's parameterized actor with a greedy actor already during training. We show that this simplification leads to faster learning, higher final performance, and reduced training time at the cost of increased inference time. While also exploring simplification of CRL through the use of a single encoder, we find that this does not allow for successful learning.

Our results provide insights into the workings of CRL and suggest that it can be further simplified and improved by focusing on its core idea of learning a critic that directly solves goal-conditioned RL problems. The implementation of our experiments is open-source and available on GitHub[1].

[1]`https://github.com/jumorisse/Analyzing-and-Simplifying-Contrastive-RL`

# Contents

**1 Introduction** **4**

**2 Related Work** **7**

**3 Preliminaries** **9**
3.1 Goal-Conditioned Reinforcement Learning . . . . . . . . . . . . . . . . 9
3.2 Contrastive Representation Learning . . . . . . . . . . . . . . . . . . 10
3.3 Contrastive Learning as Goal-Conditioned Reinforcement Learning . . 11

**4 Experiments** **15**
4.1 Analyzing Contrastive Reinforcement Learning . . . . . . . . . . . . . 15
4.1.1 Visualizing Contrastive Reinforcement Learning . . . . . . . . 15
4.1.1.1 Visualizing the Representation Space . . . . . . . . . 16
4.1.1.2 Visualizing the Critic . . . . . . . . . . . . . . . . . . 18
4.1.1.2.1 Critic Values for Varying Goals . . . . . . . . . . . . 18
4.1.1.2.2 Critic Values for Varying States . . . . . . . . . . . . 21
4.1.2 Action Selection Strategies . . . . . . . . . . . . . . . . . . . . 21
4.1.2.1 Greedy vs. Parameterized Action Selection . . . . . . 22
4.1.2.2 Alignment of Parameterized and Greedy Actors . . . 23
4.1.2.2.1 Measuring Alignment of Parameterized and Greedy Actors . . . . . . . . . . . . . . . . . . . . . . . . . 23
4.1.2.2.2 Visualizing Alignment of Parameterized and Greedy Actors . . . . . . . . . . . . . . . . . . . . . . . . . 24
4.2 Simplifying Contrastive Reinforcement Learning . . . . . . . . . . . . 26
4.2.1 Greedy Contrastive Reinforcement Learning . . . . . . . . . . 26
4.2.2 Contrastive Learning with a Single Encoder . . . . . . . . . . 28

**5 Discussion** **31**

**6 Conclusion** **34**

**7 Appendix** **39**
7.1 Spiral Maze Environment . . . . . . . . . . . . . . . . . . . . . . . . 39
7.2 Dimensionality Reduction via t-SNE . . . . . . . . . . . . . . . . . . 39
7.3 Visualizations of Greedy CRL . . . . . . . . . . . . . . . . . . . . . . 40
7.3.1 Visualizing the Greedy Representation Space . . . . . . . . . . 41
7.3.2 Visualizing the Greedy Critic . . . . . . . . . . . . . . . . . . 42

# 1. Introduction

For any intelligent actor, being able to interact with their environment requires the actor to have a comprehensive internal representation of the environment's current state. In humans, sensory stimuli of different modalities are first processed by specialized cortical regions before being fed into higher-level neural networks which integrate sensory representations into a multi-modal representation of our surroundings. Neuroscience research has further shown our brain does not only use representations specific to sensory modalities but that different parts of a modality-specific cortex also seem to produce representations for specific tasks. In the visual cortex, for example, the ventral stream processes visual stimuli for object recognition and categorization, while the dorsal stream does so for visual guidance of movement (Goodale and Milner 1992). While our understanding of human sensory processing is still limited and subject to ongoing research, it has already inspired techniques to allow artificial agents to produce better representations of their inputs. One prominent example of this are Convolutional Neural Networks which attempt to replicate our understanding of the brain's visual cortex's hierarchical approach of first recognizing basic features like edges before combining these basic into complex features like objects (Riesenhuber and Poggio 1999; Kriegeskorte 2015). Our work builds upon recent work using predictive representations, i.e. representations that contain information that allows to reason about future outcomes instead of only representing the current state. Interestingly, prior works in neuroscience suggest that such representations are also used by the human brain (Friston 2005; Rao and Ballard 1999), some even arguing they may be building blocks of intelligence of any kind (Carvalho et al. 2024).

In reinforcement learning (RL), representations are the only information agents have about their current environment. Therefore, good representations need to contain any information required to decide on the optimal next action. One straightforward approach, used in early RL work, is to manually define representations (Tesauro 1995). Given a game, for example, one could define representations as a collection of player, NPC, and item coordinates. However, defining such a collection of necessary information is not feasible for complex environments. As a result, RL research has quickly focused on techniques that learn representation automatically in an end-to-end fashion as part of the reinforcement learning process (V. Mnih et al. 2015; V. Mnih et al. 2013).

When enabling agents to learn their own representations, researchers have two options. The first one is to give the agent full autonomy in learning its representation function, focusing solely on maximizing the RL signal, i.e., the obtained rewards. This approach, called end-to-end training, uses the RL signal to optimize both the policy network and a prepended representation network (V. Mnih et al. 2013). The second option is to define a representation objective that is used either to learn representations on auxiliary tasks decoupled from the RL task (Stooke et al. 2021), or as an auxiliary loss for end-to-end training (Mazoure et al. 2020). The RL field has seen a shift towards the second option as it improves stability, and control of what information representations contain (Eysenbach et al. 2023; Liang et al. 2015). However, this approach requires defining objectives for both the RL problem and the representation learning problem. This is problematic not only because the selection and definition of auxiliary objectives is non-trivial but also because it can hinder generalization if these objectives are domain- or task-specific (Lin et al. 2019; Lyle et al. 2021).

In this work, we build upon a recently proposed approach that takes a new route. Instead of doing representation learning through end-to-end RL, or in isolation, Eysenbach et al. (2023) propose to perform only representation learning in such a way that it also directly solves goal-conditioned RL problems. This is achieved by learning representations of state-action pairs and states using contrastive learning that pushes state-action and state representations to be similar if they are likely to appear in the same trajectories and dissimilar if they are unlikely to appear together. As a result, the similarity between learned representations encodes the likelihood of reaching a goal state given a state-action pair and, thus, directly corresponds to a Q-function under the given goal state. This approach, which we call Contrastive Reinforcement Learning (CRL), does not only reduce objectives of reinforcement and representation learning into a single objective but also does not require defining a reward function. While Eysenbach et al. (2023) conduct a comprehensive performance evaluation of CRL, they do not provide an in-depth analysis of the resulting representation space and actor behavior. Further, we see potential simplifications to their approach in using representations directly for action selection and using the same encoder for both state-action and state representations. To address these points, we conduct a qualitative analysis of the approach by Eysenbach et al. (2023), validate its assumptions experimentally, and explore potential simplifications. Our aim is two-fold. First, we want to analyze CRL to better understand its learned representation space, critic function, and the resulting actor behavior. Second, we want to explore whether CRL can be simplified while maintaining its performance. Specifically, our work addresses two research questions for each of these two aims:

1. **Analyze:** What representation space and critic function does CRL learn and how can the critic be used to solve goal-conditioned RL problems?

   1.1. Do the state-action, and goal representations learned by CRL encode the structure of the given environment?

   1.2. Does CRL indeed reduce the RL problem to choosing an action whose state-action representation is closest to the goal representation?

2. **Simplify:** Can CRL be simplified?

   2.1. Can the CRL actor network be substituted with a greedy actor and, if so, how does this affect training and the final performance?

   2.2. Can CRL be simplified to use only a single encoder for both state-action pairs and goals?

We find that both, state-action and goal representations, encode the structure of the environment while occupying disjoint regions of the representation space. Further, visualizations of the critic function confirm that it indeed correlates with the probabilities of state-action pairs to lead to a given goal state. We also quantitatively confirm this finding by showing that a greedy action selection based only on actions' critic values can achieve performance equal to that of the parameterized actor used by Eysenbach et al. (2023). Following that, we show that such a greedy actor is not only able to use the learned critic function but can also be used during training to collect the experiences used to train the critic. In fact, we find that using a greedy policy during training leads to faster training and higher final performance, at least for our low-dimensional environment. Finally, we find that a simplification by using a single encoder for state-action pairs and goals did not achieve competitive results. Whether such a single-encoder approach is unfeasible or possible with more fundamental changes is a question we leave to future work.

In the following, Chapter 2 provides an overview of prior work on representation learning for reinforcement learning. Chapter 3 describes goal-conditioned reinforcement learning, contrastive representation learning, and how Eysenbach et al. 2023 relate the two to arrive at the novel CRL approach. Following that, Chapter 4 describes the experiments designed to answer our research questions and presents their results. We continue with a critical discussion of our work and point out potential directions for future work in Chapter 5. Lastly, Chapter 6 reflects on our findings and their implications.

# 2. Related Work

An agent's understanding of its environment is crucial to its ability to perform tasks within it. Thus, defining or learning good representations of environment states can make the difference between successful and unsuccessful RL. Early work on RL commonly used manually defined representation schemes that encode the information deemed relevant. The well-known backgammon algorithm TD-Gammon, for example, represents the current game state using raw information (the positions of game pieces) and higher-level features (e.g. the probability of being hit) that were computed from the raw information following manually designed rules (Tesauro 1995).

Manually defining representations requires domain knowledge and does not allow to adapt to environment changes. To overcome these limitations, RL research has seen a shift towards end-to-end representation learning (V. Mnih et al. 2013; V. Mnih et al. 2015). In such setups, a policy- or value-network takes in the raw environment (e.g. as an image) and learns to produce its own representations in the hidden layers. Because the network is optimized for maximizing the obtained rewards, it also learns to produce those representations that lead to the highest rewards.

While manually defined representations provide full control of what to represent at the cost of performance and manual work, end-to-end representation learning is automatic and achieves better performance at the cost of producing uncontrollable and uninterpretable representations. To provide more control and interpretability in end-to-end approaches, prior work has utilized auxiliary losses that are computed based on a network's hidden layers and, together with the RL objective, make up the loss for which the network is optimized (Liang et al. 2015; Jaderberg et al. 2017). Through the definition of this auxiliary loss, hidden representation can be pushed to conform to desired properties or to capture specific information (Bruin et al. 2018). Another way to allow automatic representation learning with increased control is to decouple representation from reinforcement learning (Stooke et al. 2021). Here, a separate network is trained to produce representations for an auxiliary task like image classification or reconstruction, and the resulting representations are then used to train a RL agent.

Similar to applications of contrastive learning for tasks in computer vision (T. Chen et al. 2020; He et al. 2020; Wu et al. 2018; Hjelm et al. 2019; Sermanet et al. 2018) and natural language processing (A. Mnih and Kavukcuoglu 2013; Mikolov et al. 2013; Gao, Yao, and D. Chen 2021), RL has also been shown to benefit from contrastive learning (Oord, Li, and Vinyals 2018). Since then, contrastive learning has been used for defining auxiliary representation losses (Mazoure et al. 2020; Laskin, Srinivas, and Abbeel 2020) or as an objective for decoupled representation learning (Stooke et al. 2021; Anand et al. 2019). The main appeal of using contrastive learning for RL lies in its ability to learn representations that encode temporal relations from unlabeled experiences. Once state representations contain information about their temporal correlation, this information can be used to predict the probability of reaching one state from the other. This idea has been used by prior work to learn models of the discounted state-occupancy-measure which describes a probability distribution over future states given a current state and action. Such a model can then be used together with provided success examples (Hatch et al. 2023) or a small set of reward-labeled states (Mazoure et al. 2023) to learn a policy.

Our work builds upon a recently proposed idea that goes a step further: Instead of performing representation learning as a means for RL, solve certain (i.e. goal-conditioned) RL problems directly by learning representations whose similarity encodes the Q-function and, thus, which action to take (Eysenbach et al. 2023). We refer to this approach as Contrastive Reinforcement Learning (CRL). Like some other works (most notably Hatch et al. 2023), CRL uses contrastive representation learning from experiences to approximate a goal-conditioned Q-function and use it to determine which action leads to a given goal. However, CRL does not require a collection of previously gathered experiences but, instead, learns representations from experiences collected online.

Since the proposition of CRL, additional work has explored how it can be applied to robotics (Zheng et al. 2024), how it can be extended to use off-policy experiences (Zheng, Salakhutdinov, and Eysenbach 2024), and how its learned representations can be used for planning and inference (Eysenbach et al. 2024). In our work, we focus on the originally proposed version of CRL and provide insights into its representation space, and resulting actor behavior. Further, we explore possible CRL simplifications, including a greedy CRL version that learns faster and achieves higher final performance on a low-dimensional environment.

# 3. Preliminaries

## 3.1 Goal-Conditioned Reinforcement Learning

Reinforcement learning (RL) deals with problems of sequential decision-making where an agent interacts with an environment. The agent observes the environment's current state, selects and performs an action, and receives a reward together with the environment's next state. Assuming that the environment's future only depends on the current state and no past state, such problems are commonly referred to as Markov Decision Processes (MDP). We adopt the formalization used by Silver et al. (2014) and define MDPs as a tuple of five elements: $(S, A, p, r, \gamma)$. $S$ describes the set of all possible states the environment can be in. $A$ denotes the set of all actions the agent can select. The transition function $p(s'|s, a)$ describes the probability of the next state being $s'$ when performing action $a$ in state $s$. The reward function $r(s, a)$ specifies which reward the agent gets when performing action $a$ in state $s$. The discount factor $\gamma$ lies in the range $[0, 1]$ and is used to weigh current and future rewards. Since MDPs describe sequential problems, we use an index $t$ to denote states, actions, or rewards at specific timesteps. For example, $s_0$ describes the initial state of our environment (at timestep 0) and $s_1$ describes the state at timestep 1, i.e. after performing the first action. Given an MDP, RL aims to learn a policy $\pi$ that maps states to a probability distribution over actions, from which the agent can sample to select actions. The goal of RL is to find the optimal policy $\pi^*$ that maximizes the expected cumulative reward the agent receives in the future:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\substack{\pi \\ p}} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \tag{3.1}$$

Goal-conditioned RL is a special case of RL in which we have determined a certain state $g \in S$ to be the goal that we are trying to get to. Its formalization is almost identical to that of the MDP given above with the exceptions of using a goal-conditioned reward function $r_g$ and policy $\pi_g$. While the functionality of these elements remains the same, i.e. the reward function maps state-action pairs to rewards and the policy maps states to actions, they now do so differently given different goals.

The goal-conditioned reward function can also be expressed as the probability of reaching the goal with the next step:

$$r_g(s_t, a_t) = (1 - \gamma) p(s_{t+1} = g | s_t, a_t) \tag{3.2}$$

Further, the optimal goal-conditioned policy $\pi_g^*$ and Q-function are defined as follows:

$$\pi_g^* = \arg\max_{\pi_g} \mathbb{E}_{\substack{\pi_g \\ p}} \left[ \sum_{t=0}^{\infty} \gamma^t r_g(s_t, a_t) \right] \tag{3.3}$$

$$Q^\pi(s,a) = \mathbb{E}_{\substack{\pi_g \\ p}} \left[ \sum_{t'=t}^{\infty} \gamma^{t'-t} r_g(s_{t'}, a_{t'}) | s_t = s, a_t = a \right] \tag{3.4}$$

Lastly, the discounted state-occupancy-measure describes the expected frequency with which a particular state, e.g. the goal state, is visited when following a certain policy from a starting state. It is defined as:

$$p^{\pi_g}(s_{t+} = s) = (1-\gamma) \sum_{t=0}^{\infty} \gamma^t p_t^{\pi_g}(s_t = s) \tag{3.5}$$

with $p_t^{\pi_g}(s)$ describing the probability density over states at timestep $t$ when following policy $\pi$. Further, $s_{t+}$ denotes states that were sampled from the discounted state occupancy measure.

## 3.2 Contrastive Representation Learning

The general idea of contrastive representation learning is simple: Given pairs of inputs that are marked as positive and negative pairs, a representation function should be learned that produces similar representations for positive pairs and dissimilar representations for negative pairs. Unlike traditional methods such as maximum likelihood estimation, which often focus on fitting a model to the observed data or predicting labels, contrastive learning emphasizes learning relationships between data points. Additionally, contrastive learning does not require labeled data as long as positive and negative pairs are defined in a way that captures the desired relationships. For learning word embeddings, for example, this could be done by comparing the sentences in which words occur and defining them as positive pairs if these sentences share a sufficient number of words. Because of these advantages, contrastive representation learning is widely applied in fields like computer vision and language modeling (Mikolov et al. 2013; T. Chen et al. 2020; He et al. 2020; Oord, Li, and Vinyals 2018; Wu et al. 2018; Hjelm et al. 2019; Sermanet et al. 2018).

To capture meaningful relationships between data points, contrastive representation learning aims for a representation function that optimizes mutual information. Mutual information measures how well the learned representations of positive pairs are related compared to those of negative pairs. There are several formalizations of this measure. Eysenbach et al. (2023) use the InfoMax objective (Hjelm et al. 2019). Given a first element $u$ that forms a positive pair with element $v^+$ sampled from a joint distribution $p(u,v)$ and a negative pair with element $v^-$ sampled from the product of marginal distributions $p(u)p(v)$, it is defined as:

$$\max_{f(u,v)} \mathbb{E}_{\substack{(u,v^+)\sim p(u,v) \\ v^- \sim p(v)}} \left[ \log \sigma(f(u,v^+)) + \log(1 - \sigma(f(u,v^-))) \right] \tag{3.6}$$

where $f(u,v) = \phi(u)^T \psi(v)$ is a critic function that computes the similarity between representations $\phi(u)$ and $\psi(v)$ as their inner product.

## 3.3 Contrastive Learning as Goal-Conditioned Reinforcement Learning

In the previous two subsections, we introduced the fundamentals of goal-conditioned RL and contrastive representation learning. We now focus on how Eysenbach et al. (2023) connect these two concepts to introduce a novel approach, which we refer to as contrastive reinforcement learning (CRL). CRL reframes goal-conditioned RL as a contrastive representation learning problem, where state-action and goal representations are learned in a way that their inner product directly corresponds to a goal-conditioned Q-function.

The first step towards framing goal-conditioned RL as contrastive representation learning is to express the goal-conditioned Q-function in probabilistic terms. This is done using the goal-conditioned reward function expressed as the probability of reaching the goal state $g$ in the next step (Equation 3.2) to express the Q-function (Equation 3.4) as the probability of reaching the goal state $g$ given a current state $s$ and action $a$ under the discounted state occupancy measure (Equation 3.5):

$$Q^{\pi_g}(s,a) = p^{\pi_g}(s_{t+} = g|s,a) \tag{3.7}$$

In other words, this Q-function can be understood as the probability of successfully reaching the goal state $g$ when starting from the state-action pair $(s,a)$ and following the policy $\pi_g$.

To estimate the Q-function using contrastive learning, the pairs $(u,v)$ are defined to consist of a state-action pair $u = (s_t, a_t)$ and a contrast state $v$. The state $v$ is, for positive pairs, a future state $v^+ = s_{t+N}, N > 0$ from the same trajectory as $u$ or, for negative pairs, a contrast state $v^- = s_c$ that is from a different trajectory than $u$. The state-action pairs are sampled from the replay buffer, i.e. from one of the transitions collected by our current or a previous policy. For positive pairs, sampling future states from the same trajectory ensures that the state-action pair $u$ has previously led to state $v^+ = s_f$. For negative pairs, sampling a contrast state $v^- = s_c$ from a different trajectory means that it is much less likely to be a state visited when starting with the state-action pair $u$ and following the policy after. Because of this sampling, positive and negative pairs capture the temporal correlation of state-action pairs and states, i.e., the likelihood of reaching a particular state from a given state-action pair. By using these positive and negative pairs to optimize the representation functions for $u$ and $v$ according to the contrastive learning objective (Equation 3.6), the learned representations are trained to encode these temporal correlations. Together, these representation functions $\phi(u)$ and $\psi(v)$ form the critic function $f(u,v)$:

$$f(u,v) = \phi(u)^T\psi(v) \tag{3.8}$$

Here, $\phi : SxA \mapsto \mathbb{R}^N$ is an encoder for state-action pairs (referred to as sa-encoder) and $\psi : S \mapsto \mathbb{R}^N$ is an encoder for states (referred to as g-encoder due to its later use for encoding goals). Both encoders map to the same N-dimensional representation space. Since the critic function is the inner product of state-action and goal representations, it measures their similarity. Because of that and representation similarities being trained to correspond to temporal correlations, the critic function effectively returns the likelihood of reaching a goal given a particular state-action pair. This makes the critic directly corresponding to a goal-conditioned Q-function.

It is important to note that the critic function learns to approximate the Q-function for any goal, instead of being specific to a single goal. This is due to the fact that the trajectories in our replay buffer from which we sample positive and negative pairs were generated in pursuit of differing goals. Therefore, the critic is trained using experiences collected by policies conditioned on many different goals. When used as goal-conditioned Q-function, the critic can then be conditioned on the goal by setting its second argument, the state, to be the goal state.

**Algorithm 1** CRL Training Procedure

```
1: **Input:** Maximum actor steps $S$, Maximum trajectory length $L$, Minimum replay buffer size $M$, Batch size $B$
2: **Initialize:** Replay buffer $\mathcal{D}$ filled with $M$ random transitions, policy network $\pi_\theta$, sa-encoder $\phi(s,a)$, g-encoder $\psi(s)$, critic function $C = \phi(u)^T\psi(v)$
3:
4: actor_steps $\leftarrow M$
5: **while** actor_steps $\leq S$ **do**
6:     Sample a task of $B$ tuples $(s_t, a_t, s_{t+N})$ from $\mathcal{D}$
7:     Compute critic loss using batch of $(s_t, a_t, s_{t+N})$ tuples
8:     Update critic function $C$ using the computed critic loss
9:
10:    Sample a batch of $B$ start states and goal states $(s_0, g)$ from $\mathcal{D}$
11:    For each pair $(s_0, g)$, generate a trajectory using $\pi_\theta$
12:    Add the generated trajectories to $\mathcal{D}$
13:    Compute actor loss for all states in the generated trajectories
14:    Update policy network $\pi_\theta$ using the computed actor loss
15:    actor_steps $\leftarrow$ actor_steps + length(generated trajectories)
16: **end while**
```

The CRL training procedure is outlined in Algorithm 1. It starts by using a random actor to collect 100 trajectories with 100 steps each. These 10,000 collected transitions then constitute the initial replay buffer which grows up to 1,000,000 transitions before older ones are replaced by new ones. After this initialization, CRL training follows an actor-critic approach, alternating between contrastively learning the critic function using experiences from the replay buffer (policy evaluation), updating the policy based on the critic (policy improvement), and gathering new experiences with the current policy. To train the critic, a batch of 256 positive contrastive pairs, i.e. a state-action pair and later state from the same trajectory, are sampled from the replay buffer. The negative pairs then consist of all other combinations of state-action pairs and states. Using these pairs, the critic function is optimized for the objective given in Equation 3.6. In practice, this is done by using the sa- and g-encoder to produce representations for all state-action pairs and states, and computing the inner products of all combinations of these representations, resulting in a matrix of critic values. This critic matrix is then compared against a label matrix, which can be understood as a binary matrix where one dimension corresponds to state-action pairs and the other to contrast states. The diagonal entries represent positive pairs and all other entries represent negative pairs, making

**Algorithm 2** Critic and actor losses, modified from Eysenbach et al. (2023)

```
from jax.numpy import einsum, eye
from optax import sigmoid_binary_cross_entropy

def critic_loss(states, actions, contrast_states):
  # Encode the state-action pairs
  sa_repr = sa_encoder(states, actions)
  # Encode goal state
  g_repr = g_encoder(contrast_states)
  # Compute inner product between all pairs of sa- and g-representations
  logits = einsum('ik,jk->ij', sa_repr, g_repr)
  # Critic loss: match of max inner products and binary labels
  return sigmoid_binary_cross_entropy(logits=logits, labels=eye(batch_size))

def actor_loss(states, goals):
  # Sample actions from policy network's output distribution
  actions = policy.sample(states, goals)
  # Encode pairs of given states and sampled actions
  sa_repr = sa_encoder(states, actions)
  # Encode goals
  g_repr = g_encoder(goals)
  # Compute inner product of all positive pairs of sa- and g-representations
  logits = einsum('ik,jk->i', sa_repr, g_repr)
  # Actor loss: inverse of inner product for positive pairs
  return -1.0*logits
```

the label matrix an identity matrix. Using the critic and label matrix, the critic loss is computed as the sigmoid binary cross entropy and, thus, rewards high critic values for positive pairs and low critic values for negative pairs while punishing the reverse. Since the learned critic can serve as a goal-conditioned Q-function, one might expect the policy to simply select actions greedily based on their critic values. However, Eysenbach et al. (2023) instead use the critic to train a policy implemented as a neural network. This network, parameterized by $\theta$, takes in the current state $s$ and goal state $g$ and outputs a distribution over actions $p(a|s, g)$, resulting in a stochastic policy. In practice, the policy network outputs a mean $\mu$ and a standard deviation $\sigma$, i.e. a normal distribution, for each action dimension. A full-dimensional action is then selected by sampling each of its dimensions from the respective normal distribution:

$$a \sim \pi_{g,\theta}(s) = p_\theta(a|s, g) = \mathcal{N}\left(\mu_\theta(s, g), (\sigma_\theta(s, g)^2)\right) \tag{3.9}$$

To train the policy network, a batch of 256 starting states and goal states are sampled from the replay buffer and the policy network is applied to attempt to reach the goals from the start states. The resulting trajectories are then added to the replay buffer and also used to train the policy network to select actions with high critic values, i.e., those that are likely to lead to the goal according to the current critic. The policy network's loss, also called actor loss, is defined as the negative critic values of the selected action and, thus, trains the policy to select actions that maximize the critic values for a given goal. Training follows this alternating actor-critic schema until a pre-defined amount of actor steps has been recorded. A JAX implementation of the critic and actor loss, as used by Eysenbach et al. (2023), is given in Algorithm 2.

Eysenbach et al. (2023) explore several different policy losses and contrastive learning objectives. In our work, we only discuss the ones used in the CRL implementation which we base our work on[1]. For details on further losses, objectives, and further theoretical background of CRL that includes proofs and convergence guarantees, we refer the reader to Eysenbach et al. (2023).

[1]`https://github.com/google-research/google-research/tree/master/contrastive_rl`

# 4. Experiments

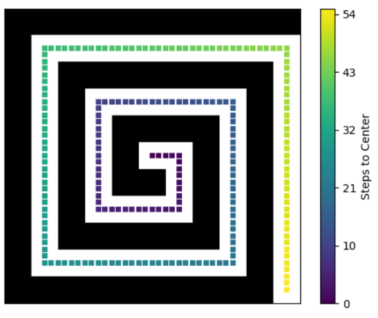
To provide a better understanding of CRL as introduced by Eysenbach et al. (2023) and test possible simplifications of this approach, we perform several experiments. Given the four research questions outlined in our introduction, the following four experiments aim to provide answers while also revealing additional insights and providing inspiration for future explorations of CRL. All experiments are performed in a 2D spiral maze environment with continuous states consisting of coordinate pairs within $[0, 11]^2$ and continuous actions within $[-1, 1]^2$. The low dimensionality of this environment simplifies visualizations and their interpretation, while its continuous states and actions ensure that our findings are not limited to overly simplistic, discrete setups. A detailed description of the environment is provided in Appendix 7.1. We start with experiments that analyze CRL, before continuing with experiments that explore possible simplifications to CRL.

## 4.1 Analyzing Contrastive Reinforcement Learning

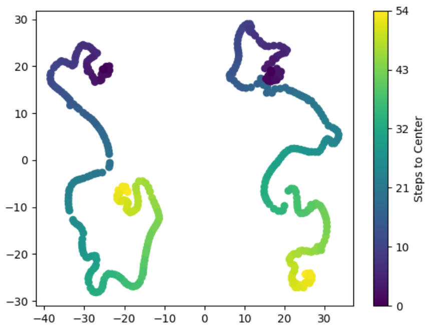
### 4.1.1 Visualizing Contrastive Reinforcement Learning

Recapping Section 3.3, the core idea of CRL can be summarized as learning state-action representations (produced by the sa-encoder $\phi : SxA \mapsto \mathbb{R}^{64}$) and goal representations (produced by the g-encoder $\psi : S \mapsto \mathbb{R}^{64}$) in such a way that their similarity in the representation space corresponds to the likelihood of reaching the goal when performing the action in the state. This implies that the learned representations map the environment's state space into the representation space in a way that captures the environment's underlying structure. After all, states close to each other in the state space are also more likely to be reached from the respective other than from one that is much further away. It is important to note that, for measuring closeness in the state space, the actionable distance matters. The actionable distance describes how many actions are necessary to reach one state from another. It can, but often does not, correspond to the metric distance in the state space. For example, two points in a maze can be only a single meter apart but, when separated by a wall, the path leading from one to the other can be much longer. In this example, the actionable distance would be much larger than the metric distance.

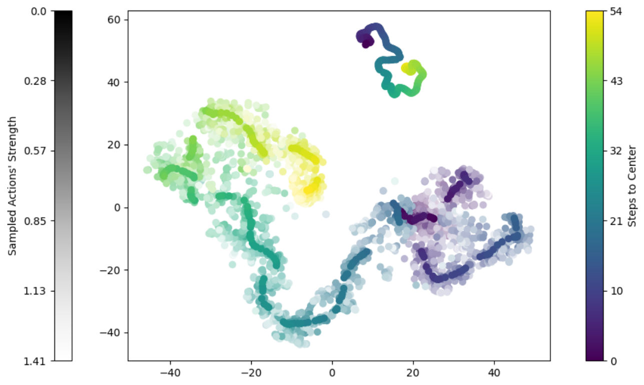
To investigate whether CRL learns representations that encode the actionable structure of the state space, we produce several latent space visualizations. Eysenbach et al. (2023) already present one such visualization that suggests the actionable structure is indeed preserved in the representation space. However, they only visualize state-action representations for fixed [0,0] actions and do not include goal



(a)



(b)



(c)

**Figure 4.1:** We visualize representations for 217 states shown in (a). (b) shows the 2D projections of the goal representations (left) and state-action representations (right) for fixed [0,0]-actions. (c) shows 2D projection when additionally including 10 state-action representations with sampled actions for each state. States and their representations are colored based on the states' step distance to the center with colors of sampled state-action pairs fading out with increasing action strengths.

representations or state-action representations for other actions. We expand their performed visualizations by addressing these limitations. Beyond that, we also visualize the critic function and how it is affected by varying actions or states.

#### 4.1.1.1 Visualizing the Representation Space

**Experiment Setup** To visualize the latent space of state-action and goal-representations, we follow the approach by Eysenbach et al. (2023). We first generate 217 states evenly spaced throughout the spiral maze as displayed in Figure 4.1a. This ensures that we reliably and evenly cover the whole state space without introducing bias into the dimensionality reduction by over- or underrepresenting certain parts of the state space. Each generated state is used to produce 12 different representations: 1 goal representation, 1 state-action representation using action [0,0] (fixed state-action representations), and 10 state-action representations with uniformly sampled actions (sampled state-action representations). Once we have produced our representations, we project them to 2D using t-SNE (van der Maaten and Hinton 2008).

However, we do so twice. Once using all $217 \cdot 12 = 2604$ representations (shown in Figure 4.1c) and once using only the fixed state-action representations together with the goal representations (shown in Figure 4.1b). This is done due to t-SNE projection being heavily influenced by the set of high-dimensional vectors that are to be projected. A more detailed explanation of this and the overall dimensionality reduction can be found in Appendix 7.2.

Once representations are projected to 2D, they can simply be plotted. To relate the state space with the representation space, we color code states and their representations. Goal representations and fixed state-action representations are colored based on the number of steps required to reach the spiral center from the respective state. Sampled state-action representations are colored based on their state's steps to the center but fade towards white the stronger the action is.

**Results** Figure 4.1b displays the 2D projections of goal and fixed state-action representations. It shows two trajectories of representations, one for goal representations (left) and one for fixed state-action representations (right). We can see that both trajectories capture the actionable structure of the environment, i.e. states which lie close in the environment also lead to representations that are close. Further, they both have a similar orientation such that representations for center states are at the top and states further from the center below. Another interesting observation is that the trajectories are never overlapping. These observations indicate that goal and state-action encoders are occupying related but disjoint regions within the representation space. Figure 4.1c displays the 2D projections when using all representations, i.e. goal representations together with fixed and sampled state-action representations. It is clearly observable that state-action and goal representations still use disjoint regions of the representation space, suggesting this is a general characteristic of the representations learned by CRL. Further, it seems like the sampled state-action representations lie around the fixed state-action representations. However, Figure 4.1c also demonstrates the drawbacks of visualizing a high-dimensional space in 2D. Compared to Figure 4.1b, the state-action representations cover a much larger space while the scale of the goal representation space has shrunk. Further, the continuous trajectory of the fixed state-action representations in Figure 4.1b is not as continuous in Figure 4.1c but a general color gradient is still visible. These differences between Figure 4.1b and 4.1c are likely to be effects of applying t-SNE different sets of representations. We further discuss how such effects limit our interpretations of such visualizations in Chapter 5 and provide a more detailed explanation of these effects in Appendix 7.2.

Overall, these visualizations of the contrastive latent space support the claim that the state-action and goal representations capture the actionable structure of the environment. All representation trajectories in Figure 4.1b and 4.1c show a color gradient consistent with the environment's structure. Additionally, the visualizations show that the state-action and goal encoders are using disjoint regions of the representation space. While effects of t-SNE limit the interpretability of these 2D projections, the consistency with which representations preserve the environment's structure and are separated into goal and state-action regions suggests that these observations are also characteristics of the high-dimensional representation space.

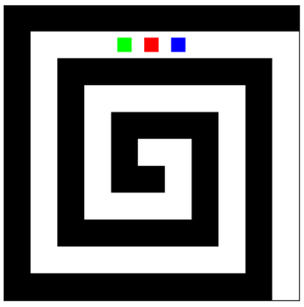
#### 4.1.1.2 Visualizing the Critic

As mentioned, the 2D visualizations of the representation space suffer from distortions caused by the dimensionality reduction technique. To overcome this issue and better understand how the distances between state-action and goal representations relate to the likelihood of reaching a goal given a state-action pair, we also visualize the critic. Since critic values are computed as the inner product between state-action and goal representations (Equation 3.7), they directly correspond to similarities in the representation space. We produce two types of visualization that aim to display how critic values are affected by varying the goal while keeping the state fixed and how they are affected when varying the state while keeping the action and goal fixed.

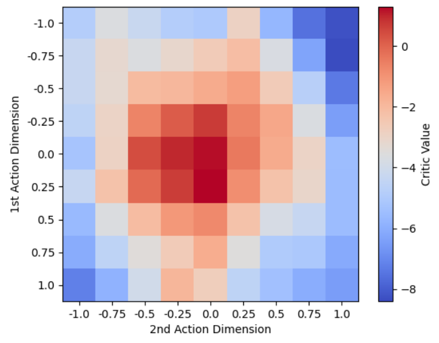
##### 4.1.1.2.1 Critic Values for Varying Goals

**Experiment Setup** The CRL critic is trained so that its values tell us which action is most likely to lead to a certain goal. To demonstrate how this looks, we define a local neighborhood, depicted in Figure 4.2a, which consists of a starting state and two neighbor states that can be reached in a single step. We always start from the red state in the middle, the green state to its left can be reached by performing action [0,-1], and the blue state to its right by performing action [0,1]. Critic values are computed and visualized for three different goal states: the start state itself, the left neighbor state, and the right neighbor state. To limit the required computations and allow for an intuitive visualization, we discretize the continuous action space into an action grid with 81 distinct and evenly distributed actions. We color action grid cells based on the critic values from blue (low values) to red (high values). Given the critic's definition and training, we expect high critic values for actions leading to or towards the goal, lower values for actions in directions orthogonal to the goal, and the lowest values for actions leading in the opposite direction of the goal.

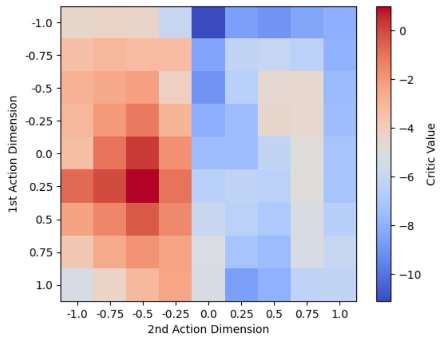
**Results** Figure 4.2b shows the critic values when using the red middle state not only as the starting state but also as the goal state. Consequently, one would expect critic values to be highest for the [0,0] action and show a gradual decrease of values based on how far a cell is from the center. The critic values shown in Figure 4.2b do indeed resemble this expectation to a certain degree. Larger values are spread out around the center where action [0,0] lies and a general gradient towards lower values at the edges is visible. Figure 4.2c shows the critic values when setting the left, green state as the goal state. Now, action [0,-1] would directly lead to the goal and, thus, should ideally have the highest critic value with a gradient from higher values on the left of the grid to lower values on the right. The values seen in Figure 4.2c do generally fulfill these expectations with some exceptions. The value peak lies around action [0.25,-0.5] and the value decline towards the right of the action grid is disrupted by actions in the second rightmost column which show higher critic values than their surrounding. Nonetheless, there is a general divide with actions leading to the left having higher values than those that lead to the right. Similar results can be observed in Figure 4.2d which shows the critic values when setting the right blue state as the goal state. The general distribution of critic values, again, shows larger critic values for actions leading towards the goal and lower values for those that don't. The value peak is again not at the ideal action [0,1] but off by a few



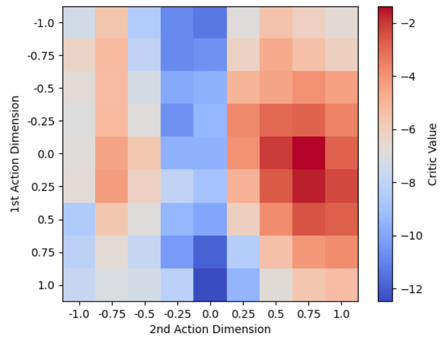
**(a)**



**(b)**



**(c)**

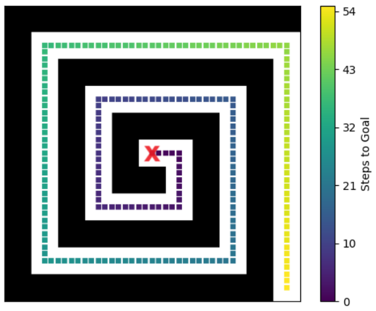


**(d)**

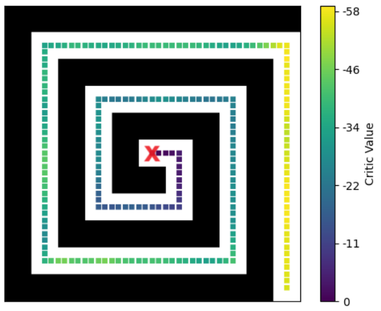
**Figure 4.2:** To visualize the critic values for different actions under varying goals, we consider a local neighborhood consisting of three states shown in (a). The middle state (red) is always used as the start state. We depict critic values for action under three different goal states: the middle state in (b), the left state in (c), and the right state in (d). Actions are colored based on their critic values.

cells and the value decline is also disrupted as it is in Figure 4.2c. We hypothesize that these disruptions are caused by trajectories in which an actor initially selected a wrong action leading away from the goal before correcting this mistake and still reaching the goal within the trajectory.

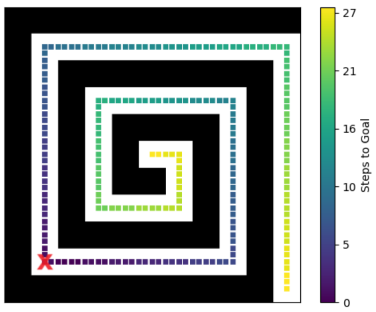
Overall, the results in Figure 4.2 show that, while not perfectly corresponding to the likelihood of reaching the goal, critic values generally point in the goal's direction. Interestingly, the critic seems better at identifying goal-oriented actions than distinguishing between actions that don't lead to the goal or away from it. Irregularities in the distribution and strength of critic values are to be expected as the critic is not trained on trajectories gathered using an optimal policy. Nevertheless, the critic's higher values for goal-directed actions support the idea that it approximates the goal-conditioned Q-function. This also supports the assumption that this learned critic can be directly used to solve goal-conditioned RL problems by choosing actions solely based on their critic value. In the following experiment, we investigate how such a greedy action selection strategy compares to the parameterized actor used by Eysenbach et al. (2023).



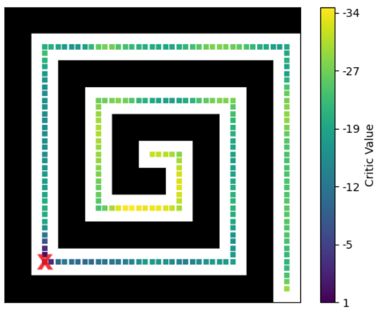
**(a)**



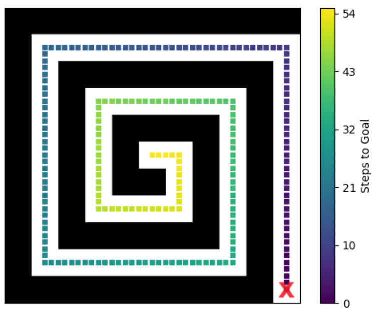
**(b)**



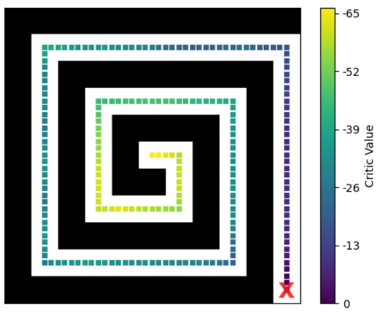
**(c)**



**(d)**



**(e)**



**(f)**

**Figure 4.3:** Critic values of various states when paired with [0,0] actions under different goals: center (top), halfway point between entrance and center (middle), and entrance (bottom). Goals are marked as X. The left plots show the states colored based on the steps required to reach the goal from each state. The right plots color states based on their critic values given the goal. In an ideal case, plots on the same row would match.

#### 4.1.1.2.2 Critic Values for Varying States

**Experiment Setup** The previous critic visualizations show that the critic function does, in general, indeed encode how likely a goal is reached given a state-action pair. This might lead one to believe that it can also be used to generally assess how close two states are in the environment. However, Figure 4.1 has already shown that state-action and goal representations are disjoint. Further, Figure 4.1b shows that, while sharing similarities, the two disjoint representation spaces do not seem to allow to reliably infer the environment distance of two states based on their goal and state-action representations. To investigate this further, we again generate the same 217 states shown in Figure 4.2a. We then produce the goal and fixed state-action representation of each of these states and use them to compute a critic value. We do this once for three different goals: the spiral center, the spiral entrance, and the state halfway between entrance and center.

**Results** Figure 4.3 displays the results. We can see that the critic values do encode the environment distance relatively well when the goal is at either the entrance or the center. When the goal is in the middle, however, critic values correspond much less to environment distances. Interestingly, this is in line with Figure 4.1b where we can see that the goal and state-action representations share a similar general orientation with states close to the center being on top and states close to the entrance being at the bottom. While this trend only holds to a certain degree, it still allows to generally infer how close a state is to the entrance or center by determining how low or high its representation is in the shown trajectory. However, when the goal is in the middle, this general rule does not hold as strong because of the trajectories' meandering form.

These findings suggest that the critic can reliably be used to decide which action in a state makes it more likely to reach a given goal, but can not be reliably used to determine how close two states are.

### 4.1.2 Action Selection Strategies

As stated in the original CRL paper *"The learned critic function not only tells us the likelihood of future states, but also tells us how different actions change the likelihood of a state occurring in the future. Thus, to learn a policy for reaching a goal state, we choose the actions that make that state most likely to occur in the future."* (Eysenbach et al. 2023, p. 5). This suggests that, once the critic function is learned, the actor can reach a goal by greedily selecting actions solely based on their critic value. In practice, however, Eysenbach et al. (2023) parameterize the actor as a neural network that takes the current state and the goal as input and outputs a distribution over actions (Equation 3.9). While the actor is optimized via gradient descent using a loss that punishes the selection of actions with low critic values (Algorithm 2), greedy action selection is never explicitly enforced. Given this mismatch between the underlying idea of CRL and its actual implementation, we want to better understand how competitive greedy action selection actually is and to which degree the parameterized actor behaves greedily. To this end, we perform two experiments. First, we evaluate whether greedy actors can indeed compete with the parameterized actor when confronted with the same tasks. Second, we evaluate and visualize to which degree actions align when chosen by different agents in the same situation.

#### 4.1.2.1 Greedy vs. Parameterized Action Selection

**Experiment Setup** Assuming that the trained critic indeed maps state-action pairs and a goal state to the likelihood of reaching that goal when performing the respective action in the state, an actor greedily choosing actions based on their critic value should be able to compete with an actor which is parameterized by a neural network trained to select greedily. To test this, we implement a simple evaluation framework that evaluates actors across the same 500 randomly sampled tasks. Tasks consist of a start state and a goal state which are both uniformly sampled from all states which are valid candidates, i.e. which are not a wall. Actors have a maximum of 100 steps per task to reach the goal. Task success is defined as coming within a Euclidean distance of two at any timestep of a trajectory. We compare six actors: the parameterized actor, four greedy actors that vary in their degree of random exploration, and a baseline actor that selects actions at random. The parameterized actor is the one used by Eysenbach et al. (2023) which consists of a neural network that takes the current state and goal as input and outputs one distribution per action dimension (Equation 3.9). The selected action is the result of sampling from these distributions. The random actor uniformly samples an action from the given action space. For the greedy actor, we discretize the action space so that it can easily identify the action with the highest critic value. Identifying the greedy action through continuous optimization is not feasible because the critic function's shape over all state-action pairs is not known but instead depends on the representations for all currently possible state-action pairs. Discretization is done using the same action grid as shown in Figure 4.2, i.e., the continuous action space is discretized into 81 distinct and evenly spaced actions. We consider four variants of the greedy actor, each with a different exploration probability $\epsilon$. For instance, a 99% greedy actor selects actions greedily 99% of the time and explores randomly 1% of the time. Greedy agents with random exploration are considered due to strictly greedy strategies being prone to get stuck in local optima. To account for random effects during training that can influence the resulting policy network and critic, which is used by the greedy actors, the results for each actor are not only averaged across the 500 tasks but also across five actor versions whose components were trained with different random seeds.

**Results** Table 4.1 shows the different actors' success rates averaged across all 100 tasks. Surprisingly, the random actor achieves a success rate of 44%. While this is in line with the evaluation implemented by Eysenbach et al. (2023), this high random performance does raise several questions regarding the environment and its success definition. We discuss these in Chapter 5. The parameterized actor achieves a success rate of 80%. The greedy actor's performance increases with increasing degrees of random exploration. The best greedy actor, who acts randomly in 10% of cases, achieves a performance of 80% and, thus, matches the performance of the parameterized actor. Given these results, we can conclude that the learned critic indeed allows greedy actors to reach performance equal to that of the parameterized. Interestingly, this is the case even when the critic is trained on experiences collected

**Table 4.1:** Average success rates and their standard deviations across five actor versions whose components (critic or policy network) were trained using different random seeds. A single seed's success rate is an average across 500 randomly sampled tasks. All values are rounded to two decimal points.

| **Actor** | **Success Rate** |
|---|---|
| Random | $0.44 \pm 0.02$ |
| Greedy, 100% | $0.69 \pm 0.03$ |
| Greedy, 99% | $0.72 \pm 0.03$ |
| Greedy, 95% | $0.77 \pm 0.02$ |
| Greedy, 90% | $0.80 \pm 0.02$ |
| Parameterized | $0.80 \pm 0.02$ |

**Table 4.2:** The Euclidean distances and cosine similarities of actions chosen by different actors in the same situation, i.e. same state and goal. All values are rounded to three decimal points.

| **Compared Actors** | **Avg. Eucl. Dist.** | **Avg. Cosine Sim.** |
|---|---|---|
| Parameterized & Parameterized | $0.000 \pm 0.000$ | $1.000 \pm 0.000$ |
| Parameterized & Greedy | $0.838 \pm 0.071$ | $0.383 \pm 0.051$ |
| Parameterized & Random | $1.249 \pm 0.030$ | $0.001 \pm 0.003$ |

by the parameterized actor. While this parameterized actor is trained to select actions greedily, this takes time. Considering that the replay buffer can contain up to 1,000 max-length trajectories before new trajectories replace old ones, the replay buffer is likely to contain a large portion of trajectories collected by a parameterized policy that did not yet learn to select actions greedily based on their critic function. In Experiment 4.2.1 we investigate how CRL training changes when using a greedy actor also to collect the experiences used to train the critic and whether this critic allows for even better evaluation performance for greedy actors.

#### 4.1.2.2 Alignment of Parameterized and Greedy Actors

While the previous experiment explores whether the learned critic function enables competitive performance of a greedy actor, it does not answer to which degree the parameterized actor makes greedy decisions. To investigate this, we implement another experiment in which the parameterized and greedy actors are confronted with the same decision situations, i.e. the same current state and goal. We then record their selected actions and compare them quantitatively. In addition, we visualize their action selection for a few selected situations.

##### 4.1.2.2.1 Measuring Alignment of Parameterized and Greedy Actors

**Experiment Setup** For the quantitative comparison, we again randomly sample tasks in the same way as described in Section 4.1.2.1. This time, however, we only sample 10 tasks. The parameterized agent then attempts to solve these tasks and each of its selected actions is recorded. In addition, the task's goal and the state of each timestep are also fed to the 100% greedy agent whose action selection is also recorded. This setup results in a collection of actions selected by the parameterized and greedy actors under the same conditions. We then compute the average Euclidean distance and cosine similarity of all collected action pairs. The Euclidean distance between two actions measures how far apart the points are that these actions transition to. The given action space is the square $[-1, 1]^2$ and, thus, the Euclidean distance between two actions can range from 0 (if actions are identical) to $2 \cdot \sqrt{2} \approx 2.828$ (if actions point to opposite corners of the square). We chose to also consider the actions' cosine similarity as this only considers the orientation of actions and does not take into account their strength. This is in line with the task definition, as success is independent of how many steps were used to reach the goal as long as it takes not more than 100 steps and, therefore, actions can be equally good if pointing in the same direction with different magnitudes. Cosine similarity ranges from -1, indicating opposing direction, to 1, indicating identical direction. To put these measures into perspective, we also include two baseline comparisons. For the first baseline, we compare parameterized with random action selection. This comparison can be understood as a lower bound of alignment due to the random selection. The second baseline is the comparison between two parameterized actors. We included this baseline due to the stochastic process with which parameterized actors select actions. Such stochasticity can lead to misalignment even when comparing two same actors which is why the comparison between two parameterized actors serves as an upper bound of alignment.

**Results** Table 4.2 shows the results of the different comparisons. The two parameterized actors reach perfect alignment scores, i.e. an average Euclidean distance of 0 and an average cosine similarity of 1. This lack of even minor misalignment suggests that the normal distributions over actions output by the policy networks have collapsed with very small standard deviations. The second baseline, the lower bound of alignment, in which a parameterized actor is compared to a random actor leads to an average Euclidean distance of 1.249 and an average cosine similarity of 0.001. These values lie around the middle of the metrics' respective value ranges. More extreme values, e.g. a cosine similarity of -1, would not indicate misalignment but rather inverse alignment, i.e. actors always choosing the opposite action. Given these lower and upper bounds of alignment, we can interpret the alignment between the parameterized and the greedy agent. Actions selected by these two agents show an average Euclidean distance of 0.838 and an average cosine similarity of 0.383. Taking the baselines into account, these scores lie closer to the lower bound of alignment than they do to the upper bound. This indicates significant misalignment between the parameterized and 100% greedy action selection. We do not consider other variants of greedy action selection because the inclusion of random choices would only lead to stronger misalignment.

#### 4.1.2.2.2 Visualizing Alignment of Parameterized and Greedy Actors

**Experiment Setup** While the prior analysis looked at action differences across a large number of samples, we also want to get a better insight by looking at a few specific cases. To do so, we selected 55 states which are evenly distributed throughout the spiral maze. Further, we set the maze's center to be the goal. Given that goal, we performed action selection for each of the 55 states and did so once using the parameterized agent and once using the greedy agent.

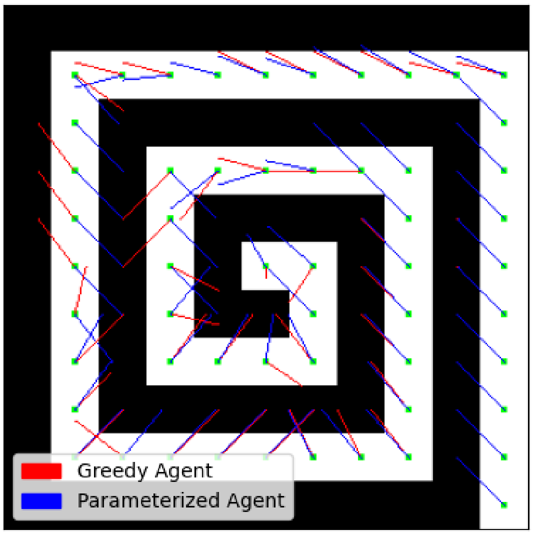


**Figure 4.4:** Actions chosen by the greedy (red) and parameterized (blue) actors at different states (green squares) when given the spiral center as the goal state. While selected actions are almost identical for states close to the entrance of the spiral, later states show very different and even opposing action selection.

**Results** Figure 4.4 visualizes the actions selected by each agent in each of the 55 states. For the considered situations, parameterized and greedy action selection show different levels of alignment. In the nine states closest to the maze entrance, parameterized and greedy actors choose practically identical actions. Action selection remains relatively similar for the following 11 states before becoming almost completely opposed in the next three states. This large variety of alignment and misalignment continues throughout the remaining states. Overall, there is no recognizable pattern of when action selections align and when they do not. Another interesting observation is that both the parameterized and greedy agent seem to always choose actions with a length close to 1, except when already having reached the goal state. This is also the case when such strong actions lead into a wall. It seems like the agents ignore the walls and, instead of reducing their actions, just rely on the environment's implementation which truncates actions that would lead into walls.

Summing up, our experiments about alignment of parameterized and greedy action selection have shown that there is a significant difference between the two. Nonetheless, greedy action selection allows for almost matching performance when using a strategy that occasionally performs random exploration to escape local minima. This similar performance is especially surprising considering the occasionally very poor greedy action selection observed in Figure 4.4. One possible explanation would be that these bad actions are the results of locally unreliable critic values that do not harm overall performance because these issues do not persist in the resulting next state.

## 4.2 Simplifying Contrastive Reinforcement Learning

### 4.2.1 Greedy Contrastive Reinforcement Learning

**Experiment Setup** The previously discussed experiments showed that the parameterized actor used in the original implementation of CRL does not learn to act greedily despite the fact that a greedy actor can use the learned critic to achieve competitive performance. This led us to hypothesize that the actor does not need to be parameterized and, instead, a greedy actor can be used already during the training procedure. Replacing a parameterized actor with a greedy actor has several advantages. It allows us to remove the actor neural network and all its associated model-, and hyperparameter. This also means no gradient has to be computed for every time the actor is optimized which can noticeably reduce the training time. This time advantage, however, depends on the number of discrete actions since the greedy actor needs to, for each action, produce a state-action representation and compute the critic value given the state-action and goal representation, before identifying which action has the highest critic value. An additional advantage is that the greedy actor has only two parameters: the set of actions and the exploration probability. A neural network, in contrast, requires the specification of many model parameters (e.g. network depth and width, activation functions, and definition of the output layer) and hyperparameters (e.g. learning rate, batch size, and loss function). Finally, we argue that our approach better captures the core idea of CRL since it performs action selection directly based on the critic, instead of using the critic to train an actor. This way, new experiences collected to update the critic also precisely correspond to the strategy encoded by the previous critic.

We defined the set of possible actions by using action grids similar to those used in Experiments 4.1.1.2 and 4.1.2.1. Given a current state and the goal, the greedy actor then evaluates each of the actions within this grid and selects the one with the highest critic value. We consider three versions of this greedy actor which differ in regards to the number of actions considered. Specifically, we consider action grids of sizes $3 \times 3$, $5 \times 5$, and $9 \times 9$. For this experiment, we do not compare greedy actors with different degrees of exploration. Instead, we use the 95% greedy actor. While our final results of Experiment 4.1.2.1 identified the 90% greedy actor as the best performing, intermediate results favored the 95% greedy actor which is why we chose this actor for this experiment. Besides replacing the actor, we preserve the original implementation and do not change any hyperparameter. Therefore, we only changed how the actor chooses actions from using a neural network to using greedy selection based on the current critic.

We train until the actor has taken 500,000 steps. As baselines, we also perform the same training using a random actor and the original CRL setup by Eysenbach et al. (2023) which uses the parameterized actor. To account for possible variations between training runs of the same setting, we perform each training procedure using five different random seeds and average results over these seeds.

Tasks used during training are sampled from the replay buffer. To evaluate the resulting actors on tasks sampled from the whole state space, we apply the evaluation setup used in Experiment 4.1.2.1. We evaluate greedy actors with a 10% probability for random exploration, differing from the 5% exploration probability used during

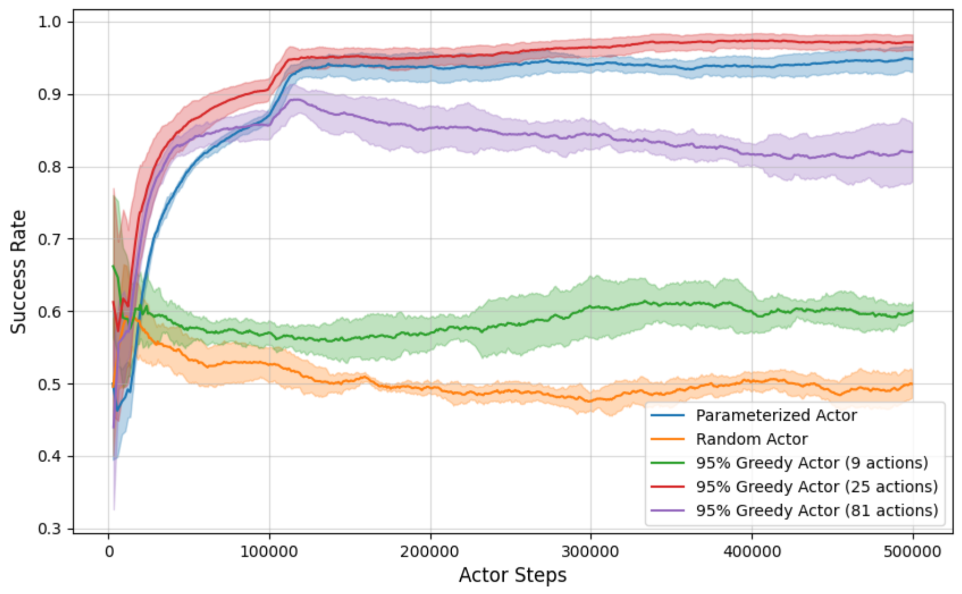


**Figure 4.5:** Success rates throughout training using the original (blue), random (yellow), and our three 95% greedy actor versions who choose from 9, 25, or 81 actions (green, red, purple). Success rates are running averages over the last 1,000 trajectories. Shown curves are the means across five random seeds with the confidence intervals showing the standard deviations across seeds.

training. This decision is motivated by the results of Experiment 4.1.2.1 in which 10% random actions led to the best performance for greedy actors.

**Results** Figure 4.5 shows the training curves of all compared variants. We can see that for two of our greedy actors, the ones selecting from 25 or 81 actions, performances increase faster than in the original setup. The setup with 81 actions does not maintain this initially high performance and ends up with a final success rate below that of the original setup. Using a greedy agent with 25 actions, however, does not only allow an initially steep learning curve but also stable performance throughout training which results in a final performance higher than that of the original setup. A greedy agent which can select from only 9 actions does not allow for competitive performance and is only slightly better than the random actor. These results indicate that the number of discrete actions considered by greedy agents has a strong influence not only on the maximum performance achieved but also on the stability of learning. Using a too-small set of actions seems to not allow the selection of certain actions which are required to reach a competitive performance. Using a too-large set of actions, on the other hand, seems to cause learning instabilities which can lead to performance decreases throughout learning. However, when the set of actions is set to the right size, CRL using a greedy actor learns faster and achieves better performance than the original setup.

Table 4.3 shows the results of evaluating the different greedy CRL versions compared to the parameterized actor and a random actor. We can see the order of evaluation performances matches that of the training performances. The greedy

**Table 4.3:** Mean success rates when evaluating the trained actors on 500 randomly sampled tasks consisting of a start and goal state. Results are averaged across five random seeds and shown with their standard deviations across seeds. All values are rounded to two decimal points.

| **Actor** | **Success Rate** |
|---|---|
| Parameterized | $0.80 \pm 0.02$ |
| Random | $0.44 \pm 0.01$ |
| Greedy (9 actions) | $0.57 \pm 0.03$ |
| Greedy (25 actions) | $0.84 \pm 0.05$ |
| Greedy (81 actions) | $0.66 \pm 0.03$ |

**Table 4.4:** Training and inference times of all actors. Training times were recorded for training with 500,000 actor steps and are averaged across 5 training runs using different random seeds. Inference times correspond to the times required to produce a 100-step trajectory and are averaged across 500 trajectories.

| **Actor** | **Training Time** (hh:mm:ss) | **Inference Time** (milliseconds) |
|---|---|---|
| Parameterized | 02:23:32 | 140 |
| Greedy (9 Actions) | 01:24:44 | 310 |
| Greedy (25 Actions) | 01:28:16 | 700 |
| Greedy (81 Actions) | 01:33:10 | 2084 |

actor choosing from a set of 25 actions performs best and achieves a success rate of 84%, outperforming the parameterized actor who achieves 80%.

Table 4.4 shows the training and inference times of each actor. We can see that the greedy actors train faster but at the cost of increased inference times. Interestingly, the greedy actors only show minor differences in training time while increasing the size of the action set leads to exponentially growing inference times.

Considering these results, we can confirm our hypothesis that CRL can be successfully trained using a greedy actor instead of a parameterized one. Further, greedy CRL, if using the right set of discrete actions, is more sample efficient and maintains a higher final training and evaluation performance. Removing the need to optimize the parameterized actor also allowed to reduce overall training time by almost 40%. However, greedy actors have a slower inference time compared to the parameterized actor.

### 4.2.2 Contrastive Learning with a Single Encoder

**Experiment Setup** The CRL method proposed by Eysenbach et al. (2023) makes use of two encoders, one for state-action pairs (the sa-encoder) and one for goals (the g-encoder). Experiment 4.1.1.1 has shown that both encoders preserve the structure of their inputs in similar ways, i.e. inputs with state/goals that are close in the state space also end up close in the representation space. In addition, we have found that sa- and g-encoder seem to be mapping into disjoint representation spaces. Nonetheless, one could imagine a setup in which both encoders map to the same space and goal representations are identical with representations of the respective

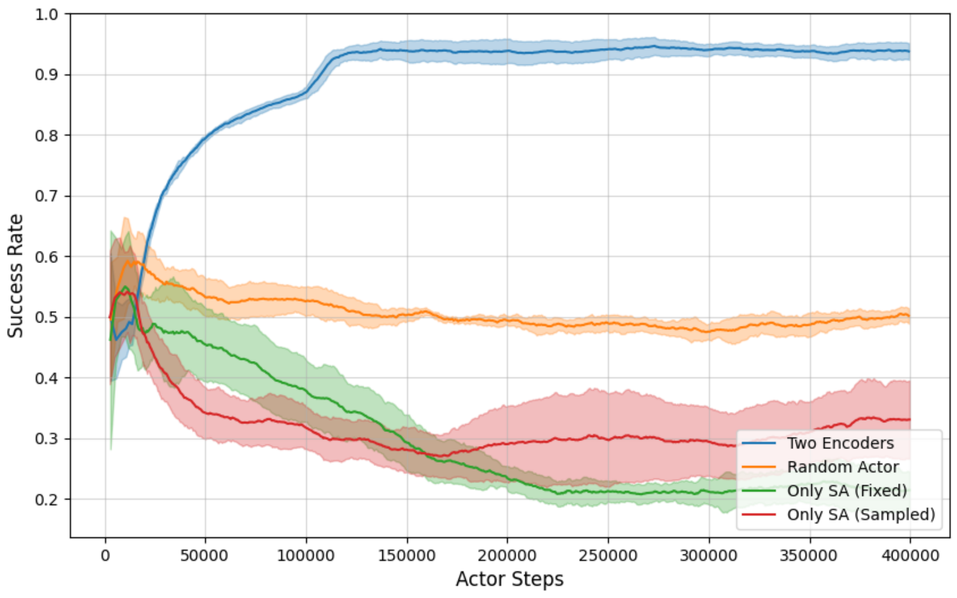


**Figure 4.6:** Success rates throughout training using the original setup (blue), a random actor (yellow), and our two single-encoder approaches that encode goals with fixed (green) or sampled (red) actions. Success rates are running averages over the last 1,000 trajectories. Shown curves are the means across five random seeds with confidence intervals showing the standard deviations across seeds.

state paired with some fixed action like [0,0], or with the representation average across all state-action pairs. Indeed, there is prior work on contrastive representation learning of images in which two encoders are trained to encode differently augmented images while sharing parameters (Laskin, Srinivas, and Abbeel 2020). Inspired by this and motivated to further simplify CRL, we experiment with performing CRL using only a single encoder.

To be able to perform action selection based on representations, the representations need to include potential actions. Therefore, CRL can not be done without a state-action encoder, meaning a single-encoder approach must use the state-action encoder to encode both state-action pairs and goals. There are several ways in which a state-action encoder can be used to produce goal representations. We test two approaches. In the first, goals are always encoded with a fixed action. Given that no action, i.e. the [0,0] action, is the action that most likely leads to the current state, a state-action representation should be closest to the state's goal representation for action [0,0]. Our results shown in Figure 4.2b show that this indeed is the case. That is why we decide, for our first approach, to produce goal representations by concatenating the goal with the [0,0] action before feeding it into the state-action encoder. The second approach is to uniformly sample an action every time a goal is encoded. This pushes a state's goal representation to be the mean of all its state-action representations. We train both approaches on the spiral environment for a maximum of 400,000 actor steps and their performances to the original implementation by Eysenbach et al. (2023) and a random actor.

**Results** Figure 4.6 displays the results which show that neither of our approaches for training with a single encoder shows successful learning. In fact, our single-encoder setups perform worse than the random actor. Therefore, we can conclude that our considered approaches of simplifying CRL to use only a single encoder network are unsuccessful. Whether this means a single-encoder CRL is generally not possible or what other reasons could have caused our approach to fail, is discussed in the following chapter.

# 5. Discussion

The experiments and results presented in the previous section provide insights into the workings of CRL, potential simplifications, and how one such simplification leads to faster training and higher performance. However, our work also contains a failed single encoder simplification and is subject to some limitations. In the following, we discuss possible explanations for the failed single-encoder approach, point out general limitations, and discuss if and how these limitations affect the interpretation of our results. Following that, we provide a short outlook on how future work can address our shortcomings and go beyond our work.

A considerable shortcoming of our work is the failed single encoder simplification of CRL. We see several possible reasons for this with the most obvious being a lack of thorough hyperparameter tuning, a limitation we further discuss below. Beyond that, failure might be rooted in other methodological or conceptual flaws of our approach. Experiment 4.1.1 revealed that the state-action and goal encoder of the original CRL setup use disjoint regions of the representation space. Encoding goals and state-action pairs with the same encoder, limits goal and state-action representations to use the same region of the representation space and may have prevented to capture features distinct to each type of representation. Another possible reason is that single-encoder CRL might require more fundamental changes to the original framework, such as modifying the loss function or introducing a special token for goal encoding instead of using fixed or sampled actions. Lastly, recent works that build up on CRL provide more detailed reasoning for using two different encoders (Eysenbach et al. 2024; Myers et al. 2024). They argue that encoding RL states must allow for asymmetrical representations for them to be able to encode asymmetric properties of the environment. Their given example is that it is more difficult to climb the peak of a mountain from its foot than it is to slide from the peak to the foot. Using our single encoder approach, the critic value would be the same no matter whether the peak would be the goal and the foot the start or the reverse. While this makes sense, we argue that such asymmetric actionable distances do not exist in our environment and, therefore, a single encoder approach should still be possible. In conclusion, we find that simplifying CRL to use only one encoder can not be achieved through the minor changes we make to the original setup.

The primary limitation of our work lies in the use of only a single environment for training and evaluation. This is especially problematic since we use a relatively simple environment in which both action and state space are continuous but only two-dimensional. On top of that, trajectory success is defined as any state of the trajectory being within a Euclidean distance of two, allowing to reach goals even through walls (as explained in Appendix 7.1). The random actor's success rate of around 44% demonstrates that this environment does not pose a significant challenge. While we adopted this environment and its success definition directly from

Eysenbach et al. (2023), their work tested CRL on a diverse set of environments, including complex 3D robotics tasks with higher-dimensional state and action spaces, as well as image-based environments where states consist of image encodings rather than coordinates. Additionally, they also use offline setups where experiences used for contrastive learning are not gathered by an actor but come from a previously collected set of trajectories. Limiting our work to the spiral maze environment allows for intuitive visualizations and computationally efficient experimentation, but restricts us to draw conclusions only for this specific environment or, at most, for environments that show similar characteristics, i.e. low dimensional continuous action and state spaces. This limitation is especially relevant to the results obtained in Experiment 4.2.1. While our greedy CRL approach achieves faster learning and improved performance at reduced training time, we saw that these improvements depend on the set of actions a greedy actor can choose from. Environments with high-dimensional action spaces may pose significant challenges for the greedy CRL approach, as the curse of dimensionality causes the number of value computations needed for a greedy choice to grow exponentially with each additional dimension. In such cases, the increased computational cost could outweigh the advantage of bypassing the optimization of a parameterized actor. Furthermore, we observed that increasing the number of possible actions in greedy CRL can lead to unstable performance, a problem likely to worsen in higher-dimensional environments. For discrete environments, although not being considered in our work, we do not expect additional challenges. In fact, greedy CRL might be even more suitable for such environments, as no discretization is necessary and the action and state spaces are typically lower-dimensional. Validating our assumptions about how greedy CRL translates to different environments is an interesting avenue for future work.

Another limitation of our work relates to our aim of providing a better understanding of CRL's high-dimensional representation space via 2D visualizations presented in Experiment 4.1.1.1. To produce such visualizations of high-dimensional spaces requires the use of dimensionality reduction techniques like t-SNE. The problem with such techniques is that they are imperfect. It is generally not possible to map a high-dimensional space to 2D while preserving all relations of the original space. Therefore, one has to be careful not to confuse 2D observations that are artifacts of the dimensionality reduction with those that actually depict properties of the high-dimensional space. To account for that, we focus only on high-level characteristics in the 2D visualizations and do not attempt to interpret low-level features like the distance between two representations. High-level characteristics are, for example, the fact that state-action and goal representations occupy disjoint areas of the representation space and that both are arranged in an order that is aligned with the actionable structure of the environment. A detailed explanation of how we project representations to 2D and a further discussion of other limiting factors like the effects of projection hyperparameters are discussed in Appendix 7.2.

As a last significant limitation, we did not perform thorough hyperparameter tuning due to limited time and computational resources. This is the case for the single encoder approach explored in Experiment 4.2.2, and the greedy actors used in Experiments 4.1.2 and 4.2.1. For the latter, we did experiment with different values for the exploration probability and size of the action set. However, we only considered three values each and evaluated these hyperparameters in isolation. Therefore, further improvements of greedy CRL might be achievable with a more thorough tuning regime. The same holds for the single-encoder approach. Here, we did not perform any hyperparameter tuning and only compared the use of fixed versus sampled actions to encode a goal with the state-action encoder. Considering that our proposed single-encoder approach makes changes to the core functionality of CRL, a thorough hyperparameter tuning would have been warranted to allow for reliable insights into whether such a single-encoder version of CRL can be successful. Similar to preventing thorough hyperparameter tuning, our limited resources also led us to train our greedy CRL approach using a 95% greedy actor despite the final results of Experiment 4.1.2.1 showing that a 90% greedy actor performs significantly better. The choice for using the 90% greedy actor was based on preliminary results of Experiment 4.1.2.1 and it was not possible to re-run the training of greedy CRL at a later time. Nonetheless, our greedy CRL learned a critic function that allowed a 90% greedy actor to outperform original CRL on evaluation tasks. This leads us to believe also using the 90% greedy actor during training of greedy CRL might allow for even more performance gains.

Regarding avenues to extend our work, we are especially curious to see how our proposed greedy CRL approach would fare in more complex environments like the one used by Eysenbach et al. (2023). Besides that, we still see potential in a continued pursuit of simplifying CRL through the use of a single encoder. Achieving such a simplification would be especially valuable because it would be independent of that achieved via greedy CRL. Therefore, a combination of both ideas into a single-encoder greedy actor approach could potentially allow for improvements beyond those achieved by greedy CRL. Our failed experiment on using a single-encoder approach should not be understood as proof of such an approach being infeasible, especially given that we did not try any fundamental changes to the original setup and did not perform any hyperparameter tuning. We still think a single-encoder approach is generally feasible and future work could explore it by addressing the discussed possible reasons for failure. Besides addressing the limitations of our work, we see large potential in extending CRL to problems beyond goal-conditioned RL, as already pointed out by Eysenbach et al. (2023). It would be interesting to see how CRL can be applied to general RL problems where no goal states are defined. Inspiration for possible approaches is already given in prior work where a learned model of the state occupancy measure is used together with a small set of reward-labeled states to estimate the likelihoods of reaching the labeled states. By weighing reward labels with these likelihoods a Q-function can be estimated (Mazoure et al. 2023). For now, we remain with our work as a contribution to better understanding the representation space and critic function of CRL, and to the improvement through simplification of CRL for low dimensional environments.

# 6. Conclusion

In this work, we expanded on the approach proposed by Eysenbach et al. (2023), who reframe goal-conditioned RL as a problem of contrastive representation learning. Our primary focus was to analyze this method's representation space and its use as a critic function that predicts the likelihood of reaching a goal state from a given state-action pair. Through visualizations, we found the state-action and goal encoders to use disjoint parts of the representation space, while successfully capturing the actionable structure of the environment. Further visualizations revealed that the critic function does indeed allow to discriminate actions that lead towards a given goal. A quantitative evaluation revealed that a greedy use of this critic function allows for performance competitive to that of a parameterized actor which has to first learn to select actions based on their critic values. Additionally, we found a considerable misalignment in the action selection behavior of the trained parameterized actor and the greedy actor. Inspired by finding equal performance for greedy action selection, we aimed to simplify CRL by substituting the parameterized actor with a greedy actor already during CRL training. This allowed for faster training and higher final performance. Thus, this approach not only simplifies CRL by removing a neural network and its optimization but also improves CRL, at least for low-dimensional environments. A second attempt towards simplifying CRL was to merge the state-action and goal encoders into one single encoder that is used to produce both state-action and goal representations. However, this single-encoder approach was not successful and did not allow for performance beyond that of a random actor. We believe our work provides a more nuanced understanding of CRL and demonstrates that the learned critic can indeed be used directly to select actions. In fact, such a direct use of the critic can simplify and improve CRL. We hope our work inspires further research on approaching RL as a problem of contrastive representation learning.

# Bibliography

Anand, Ankesh, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R. Devon Hjelm (2019). "Unsupervised State Representation Learning in Atari". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada.* Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 8766–8779.

Bruin, Tim de, Jens Kober, Karl Tuyls, and Robert Babuška (2018). "Integrating State Representation Learning Into Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 3.3, pp. 1394–1401. DOI: 10.1109/LRA.2018.2800101.

Carvalho, Wilka, Momchil S. Tomov, William de Cothi, Caswell Barry, and Samuel J. Gershman (2024). *Predictive representations: building blocks of intelligence.* arXiv: 2402.06590 [cs.AI].

Chen, Ting, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton (2020). "A Simple Framework for Contrastive Learning of Visual Representations". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event.* Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 1597–1607.

Eysenbach, Benjamin, Vivek Myers, Ruslan Salakhutdinov, and Sergey Levine (2024). *Inference via Interpolation: Contrastive Representations Provably Enable Planning and Inference.* arXiv: 2403.04082 [cs.LG].

Eysenbach, Benjamin, Tianjun Zhang, Ruslan Salakhutdinov, and Sergey Levine (2023). *Contrastive Learning as Goal-Conditioned Reinforcement Learning.* arXiv: 2206.07568 [cs.LG].

Friston, Karl (2005). "A theory of cortical responses". In: *Philosophical transactions of the Royal Society B: Biological sciences* 360.1456, pp. 815–836.

Gao, Tianyu, Xingcheng Yao, and Danqi Chen (2021). "SimCSE: Simple Contrastive Learning of Sentence Embeddings". In: *CoRR* abs/2104.08821. arXiv: 2104.08821.

Goodale, Melvyn A and A David Milner (1992). "Separate visual pathways for perception and action". In: *Trends in neurosciences* 15.1, pp. 20–25.

Hatch, Kyle, Benjamin Eysenbach, Rafael Rafailov, Tianhe Yu, Ruslan Salakhutdinov, Sergey Levine, and Chelsea Finn (2023). "Contrastive Example-Based Control". English (US). In: *Proceedings of Machine Learning Research* 211. Publisher Copyright: © 2023 K. Hatch, B. Eysenbach, R. Rafailov, T. Yu, R. Salakhutdinov, S. Levine & C. Finn.; 5th Annual Conference on Learning for Dynamics and Control, L4DC 2023 ; Conference date: 15-06-2023 Through 16-06-2023, pp. 155–169. ISSN: 2640-3498.

He, Kaiming, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick (2020). "Momentum Contrast for Unsupervised Visual Representation Learning". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020.* Computer Vision Foundation / IEEE, pp. 9726–9735. DOI: `10.1109/CVPR42600.2020.00975`.

Hjelm, R. Devon, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Philip Bachman, Adam Trischler, and Yoshua Bengio (2019). "Learning deep representations by mutual information estimation and maximization". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.* OpenReview.net.

Jaderberg, Max, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu (2017). "Reinforcement Learning with Unsupervised Auxiliary Tasks". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net.

Kriegeskorte, Nikolaus (2015). "Deep neural networks: a new framework for modeling biological vision and brain information processing". In: *Annual review of vision science* 1, pp. 417–446.

Laskin, Michael, Aravind Srinivas, and Pieter Abbeel (2020). "Curl: Contrastive unsupervised representations for reinforcement learning". In: *International conference on machine learning.* PMLR, pp. 5639–5650.

Liang, Yitao, Marlos C Machado, Erik Talvitie, and Michael Bowling (2015). "State of the art control of atari games using shallow reinforcement learning". In: *arXiv preprint arXiv:1512.01563.*

Lin, Xingyu, Harjatin Singh Baweja, George A. Kantor, and David Held (2019). "Adaptive Auxiliary Task Weighting for Reinforcement Learning". In: *Neural Information Processing Systems.*

Lyle, Clare, Mark Rowland, Georg Ostrovski, and Will Dabney (2021). *On The Effect of Auxiliary Tasks on Representation Dynamics.* arXiv: `2102.13089 [cs.LG]`.

Mazoure, Bogdan, Remi Tachet des Combes, Thang Doan, Philip Bachman, and R. Devon Hjelm (2020). "Deep Reinforcement and InfoMax Learning". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.* Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin.

Mazoure, Bogdan, Benjamin Eysenbach, Ofir Nachum, and Jonathan Tompson (2023). "Contrastive Value Learning: Implicit Models for Simple Offline RL". In: *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA.* Ed. by Jie Tan, Marc Toussaint, and Kourosh Darvish. Vol. 229. Proceedings of Machine Learning Research. PMLR, pp. 1257–1267.

Mikolov, Tomás, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean (2013). "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.* Ed. by Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, pp. 3111–3119.

Mnih, Andriy and Koray Kavukcuoglu (2013). “Learning word embeddings efficiently with noise-contrastive estimation”. In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.* Ed. by Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, pp. 2265–2273.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis (Feb. 2015). “Human-level control through deep reinforcement learning”. In: *Nature* 518, pp. 529–33. DOI: 10.1038/nature14236.

Myers, Vivek, Chongyi Zheng, Anca Dragan, Sergey Levine, and Benjamin Eysenbach (2024). *Learning Temporal Distances: Contrastive Successor Features Can Provide a Metric Structure for Decision-Making.* arXiv: 2406.17098 [cs.LG].

Oord, Aäron van den, Yazhe Li, and Oriol Vinyals (2018). “Representation Learning with Contrastive Predictive Coding”. In: *CoRR* abs/1807.03748. arXiv: 1807.03748.

Rao, Rajesh PN and Dana H Ballard (1999). “Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects”. In: *Nature neuroscience* 2.1, pp. 79–87.

Riesenhuber, Maximilian and Tomaso Poggio (1999). “Hierarchical models of object recognition in cortex”. In: *Nature neuroscience* 2.11, pp. 1019–1025.

Sermanet, Pierre, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, and Sergey Levine (2018). “Time-Contrastive Networks: Self-Supervised Learning from Video”. In: *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018.* IEEE, pp. 1134–1141. DOI: 10.1109/ICRA.2018.8462891.

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller (2014). “Deterministic policy gradient algorithms”. In: *International conference on machine learning.* Pmlr, pp. 387–395.

Stooke, Adam, Kimin Lee, Pieter Abbeel, and Michael Laskin (2021). “Decoupling representation learning from reinforcement learning”. In: *International Conference on Machine Learning.* PMLR, pp. 9870–9879.

Tesauro, Gerald (Mar. 1995). “Temporal difference learning and TD-Gammon”. In: *Commun. ACM* 38.3, pp. 58–68. ISSN: 0001-0782. DOI: 10.1145/203330.203343.

van der Maaten, L.J.P. and G.E. Hinton (2008). “Visualizing High-Dimensional Data Using t-SNE”. English. In: *Journal of Machine Learning Research* 9.nov. Pagination: 27, pp. 2579–2605. ISSN: 1532-4435.

Wu, Zhirong, Yuanjun Xiong, Stella X. Yu, and Dahua Lin (2018). “Unsupervised Feature Learning via Non-Parametric Instance Discrimination”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018.* Computer Vision Foundation / IEEE Computer Society, pp. 3733–3742. DOI: 10.1109/CVPR.2018.00393.

Zheng, Chongyi, Benjamin Eysenbach, Homer Walke, Patrick Yin, Kuan Fang, Ruslan Salakhutdinov, and Sergey Levine (2024). *Stabilizing Contrastive RL: Techniques for Robotic Goal Reaching from Offline Data.* arXiv: `2306.03346 [cs.LG]`.

Zheng, Chongyi, Ruslan Salakhutdinov, and Benjamin Eysenbach (2024). *Contrastive Difference Predictive Coding.* arXiv: `2310.20141 [cs.LG]`.

# 7. Appendix

## 7.1 Spiral Maze Environment

All our experiments make use of the spiral maze environment as implemented by Eysenbach et al. (2023). This environment is an $11 \times 11$ square spiral path that is bound by walls. While appearing like a discrete grid-world environment, the spiral maze environment is actually continuous. States consist of XY coordinates from within $[0, 11]^2$ and actions lie within the square $[0, 1]^2$. Tasks consist of a start and a goal state which are independently and uniformly sampled from all coordinates that are not walls. The environment does not terminate trajectories once a goal is reached but, instead, always lets the actors reach the maximum trajectory length. Task success is defined as reaching within a Euclidean distance of two at any timestep of a trajectory. This is problematic as it allows to reach goals through walls, see Figure 7.1. When sampling 10,000 tasks, 8.8% of them have a start state and goal state that already fulfill the success criteria. It is due to these reasons that a random actor can achieve a success rate of around 44% in this environment.

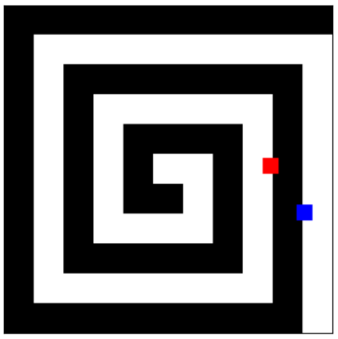
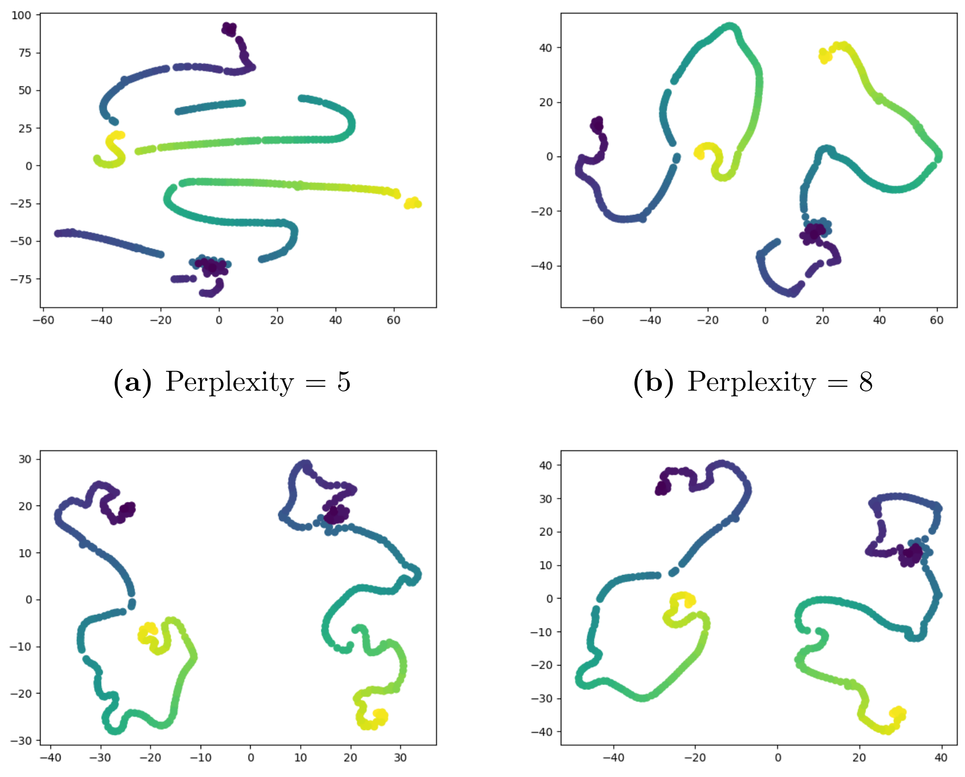


**Figure 7.1:** The spiral maze environment with a spiral path (white) and walls (black). A randomly sampled task is displayed. Despite start state (blue) and the goal (red) being separated by a wall, this task would be classified as successfully solved as the Euclidean distance between start state and the goal is less than 2.

## 7.2 Dimensionality Reduction via t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a technique for dimensionality reduction introduced by van der Maaten and Hinton (2008). The core idea behind t-SNE is to create two probability distributions that model the similarities of data point pairs in the high-dimensional and low-dimensional spaces and then minimize the difference between these two distributions. In the original high-dimensional space, the similarities between data points are modeled using a Gaussian distribution. The projection to the low-dimensional space is done using a t-distribution

As seen in Figure 4.1, the result of the t-SNE projection is strongly affected by the set of high-dimensional data points. Another factor that has a strong influence on the resulting projection is the perplexity parameter. The perplexity controls the balance between preserving relations of data points locally versus globally by determining the neighborhood of each data point. A smaller perplexity value is better suited to preserve local structures of the high-dimensional space while a larger value better preserves its global structure. Figure 7.2 shows how different perplexity values influence the resulting visualization for a fixed set of state-action and goal representations.



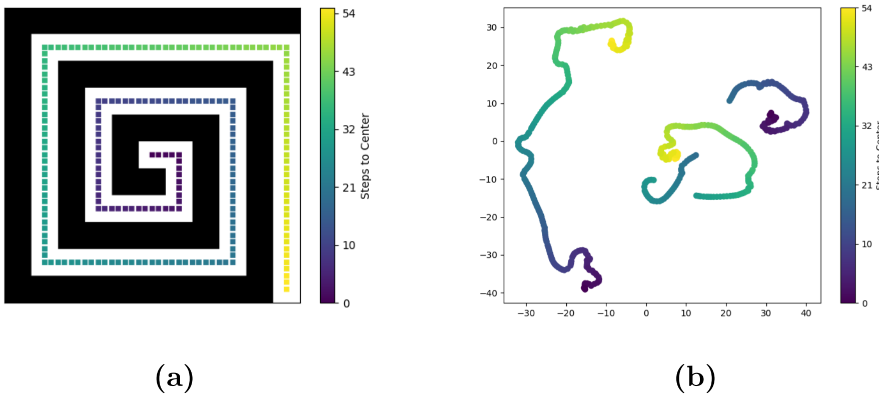
**(a)** Perplexity = 5 **(b)** Perplexity = 8

**(c)** Perplexity = 10 **(d)** Perplexity = 12

**Figure 7.2:** 2D projections of the same set of state-action and goal representations produced by t-SNE using different perplexity values of 5 (a), 8 (b), 10 (c), and 12 (d).

## 7.3 Visualizations of Greedy CRL

As we have demonstrated, greedy CRL (with a $5 \times 5$ action grid) is able to learn faster and achieve higher final performance. To evaluate whether it also leads to significant changes in the resulting representation space or critic function, we produce the same visualizations shown for original CRL in Figures 4.1 and 4.2. We do so using the same settings with the exception of using the $5 \times 5$ action grid used by the best-performing greedy CRL setting.



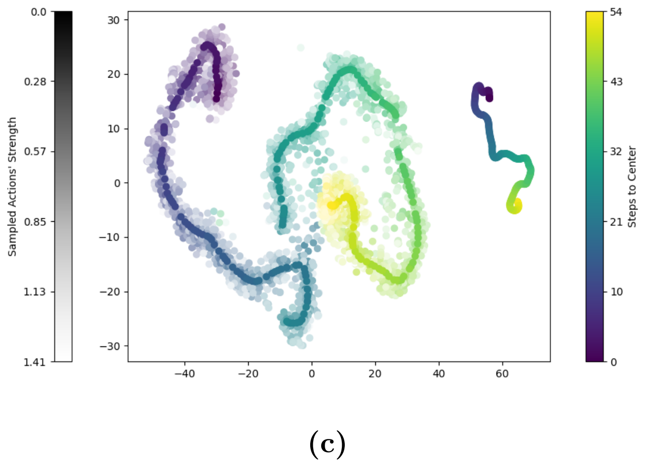


**Figure 7.3:** Visualizations of the representation space learned by greedy CRL (25 actions). We visualize representations for 217 states shown in (a). (b) shows the 2D projections of the goal representations (left) and fixed state-action representations (right). (c) shows the 2D projections when including sampled state-action representations.

### 7.3.1 Visualizing the Greedy Representation Space

Figure 7.3 shows the visualizations of the representation space learned by greedy CRL. Figure 7.3b displays the representation of goal and fixed state-action representations. Similarly to the visualization for original CRL shown in Figure 4.1b, the greedily learned representations also lie on a trajectory that is consistent with the actionable structure of the environment. For the fixed state-action pairs on the right, however, this trajectory is not continuous but rather separated into three disconnected parts which in itself are consistent. Figure 7.3c shows the visualizations of goal, fixed state-action, and sampled state-action representations learned by greedy CRL. This visualization also shares the characteristics observed for original CRL in Figure 4.1c.

In conclusion, the sa- and g-encoder trained via greedy CRL make use of disjoint regions of the representation space but learn to produce representations that capture the actionable structure of the environment. Minor differences to the visualizations of original CRL, like the discontinuous trajectory of fixed state-action representations in Figure 7.3b, might be due to using the t-SNE parameter tuned for the visualization of original CRL.

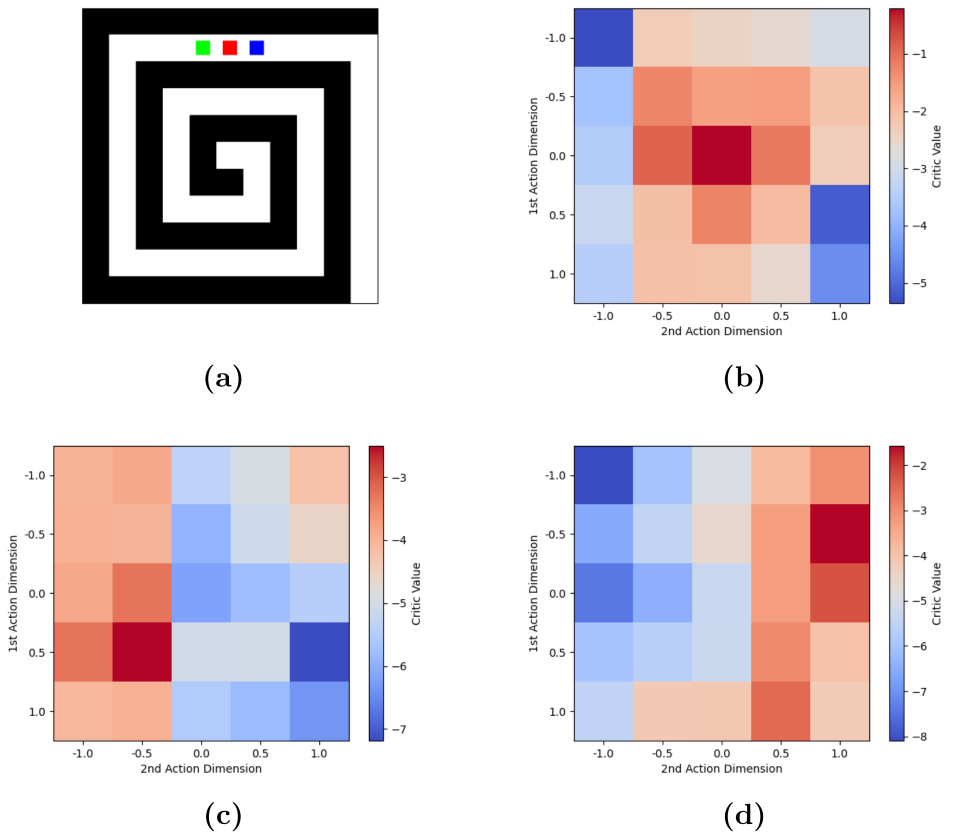


**Figure 7.4:** Visualizations of the critic function learned by greedy CRL (25 actions). We consider a local neighborhood consisting of three states (a). The middle state (red) is always used as the start state. We depict critic values of state-action pairs for three different goal states: the middle state in (b), the left state in (c), and the right state in (d). Action cells are colored based on their critic values.

## 7.3.2 Visualizing the Greedy Critic

Figure 7.4 shows the visualizations of the greedy critic using the same neighborhood used for Figure 4.2. Again, the results show similar characteristics as those for original CRL. In Figure 7.4b, for which the start state is also the goal state, we can see the highest value for action [0,0] with a general decline for cells the further they are from the center. In Figure 7.4c, where the goal is to the left of the start state, goals on the left half of the grid have higher values than those on the right. Figure 7.4d shows the critic values when the goal is to the right. Actions on the right half have higher values than those on the left. Like for original CRL, the critic trained via greedy CRL also allows to identify actions that lead towards the goal.

However, also similar to original CRL, the critic values do not perfectly correspond to the expectations one would have of a critic which perfectly encodes the likelihood of reaching a goal given a state-action pair. The optimal actions don't have the highest value, with the exception of Figure 7.4b, and the gradual decline of critic values from actions leading towards the goal to actions leading away from the goal is not smooth but rather noisy.

In conclusion, the visualizations of the critic trained via greedy CRL show the same characteristics as observed for the critic trained via original CRL.